

ドメインフレームワークについて

ドメインスペシフィックなモデル(DSM)からコードを生成させるコツは、アプリケーション内から共通した処理や関数を抽出してドメインフレームワークの部品にすることです。そうすることで、モデルから生成させるコードは、部品を呼び出す為の処理や条件判断文のみで済むようになり、ジェネレータの設定(スクリプト)は、以下のようになります。

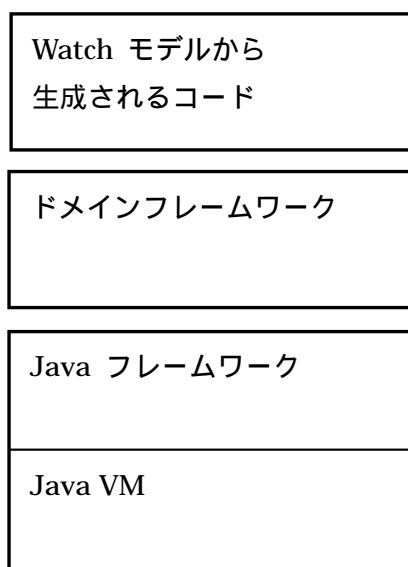
- 共通の処理、関数をメタモデルに定義されたルールや条件などに従って呼び出す
- 呼び出す時のルールや条件はパラメータとして共通の処理、関数に引き渡す
- モデル内で定義されるモデル部品のパラメータも、そのまま共通の処理、関数に引き渡す
- モデルの意図に沿ってコードとして構成する

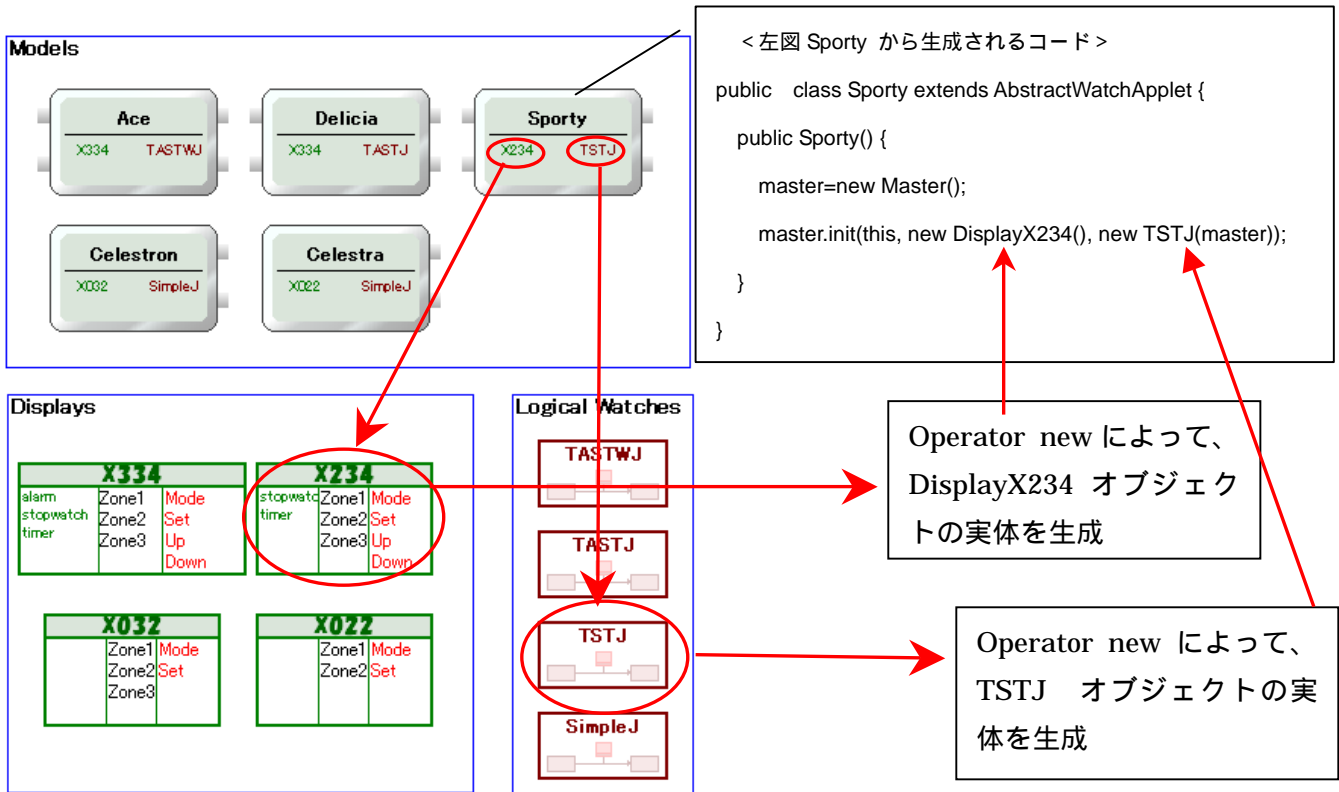
結果、アプリケーション内の似通ったコード領域は、モデル部品とそのパラメータの組合せで包括されます。そのため細かいモデル部品であふれてしまうことは有りません。コードの実装に比較して抽象度が上がり、作業効率は飛躍的に向上します。同じコードを繰り返し記述する単純作業からも開放されます。同時に、モデリング環境にルールやコツが包括されているため、コーディングミスや単純なバグ、ライブラリやコンポーネントの誤用を回避する効果も得られます。

以下インストール内の Watch Example (デジタル時計) で、モデルから生成させるコード(部品を呼び出す為の処理や、条件判断文)と、ドメインフレームワークの役割について簡単に説明します。

(注: コード生成動作に関しては、別途、“ジェネレータの動作について”をご参考下さい)

Watch Example は、“Watch モデルから生成されるコード”、“Java VM/ Java フレームワーク”、“ドメインフレームワーク”の3層構造で、デジタル時計を Java VM 上で実現します。





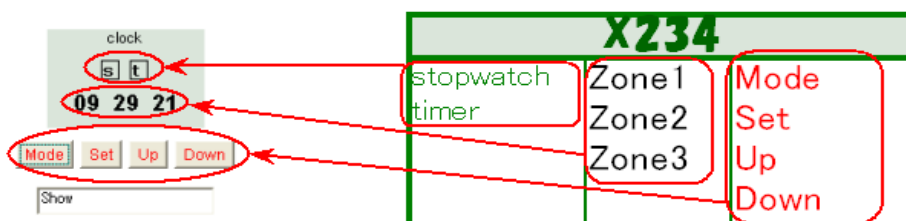
上図 Sporty(スポーツ時計)モデルには、部品としてディスプレイモデル X234 と、時計機能モデル TSTJ を組み込んでいます。この Sporty から生成されるコードは X234 と TSTJ に対するオブジェクト DisplayX234() と TSTJ(master) の生成を new で行うだけのものです。

```
public class Sporty extends AbstractWatchApplet {
    public Sporty() {
        master=new Master();
        master.init(this, new DisplayX234(), new TSTJ(master));
    }
}
```

この X234 は、時計の画面を定義するモデルです。これを実現する為に、“表示領域確保”、“データの参照”などの処理のコードをドメインフレームワーク部品として用意しました。これらについて以下に解説します。

Sporty

Watch Test Environment



例えば、ディスプレイモデル X234 上の Zone 1,2,3 は、時、分、秒のための表示領域の設定です。これに対し、モデルから生成される以下 DisplayX234() の中身は、表示領域を確保するコンポーネントを呼び出して実現しています。

Zone1 から生成されるコード `times.addElement(new Zone("Zone1"))`; は、

```
public class DisplayX234 extends AbstractDisplay
{
    public DisplayX234()
    {
        icons.addElement(new Icon("stopwatch"));
        icons.addElement(new Icon("timer"));

        times.addElement(new Zone("Zone1"));
        times.addElement(new Zone("Zone2"));
        times.addElement(new Zone("Zone3"));

        buttons.addElement("Mode");
        buttons.addElement("Set");
        buttons.addElement("Up");
        buttons.addElement("Down");
    }
}
```

X234		
stopwatch	Zone1	Mode
timer	Zone2	Set
	Zone3	Up
		Down

`times.addElement(new Zone("Zone1"))`;

(`times` は、ドメイン固有に書いたコード => Zone の表示領域確保するためのオブジェクト)

(`addElement` は、Java の標準関数 => エLEMENTの追加)

の処理を行い、結果として実行されると Zone1 の表示領域 (エLEMENT) が確保されます。

ここで `times` は Zone の表示領域を確保するための可変長配列変数です。ソースコードの実装では、このような配列変数の確保を行う処理はプログラマごとに個々にコーディングされたりしますが、この DSM の例では、`extends` を使って、以下の基本クラス (`AbstractDisplay`) を継承することで処理を実現しています。

```
abstract class AbstractDisplay
{
    Vector icons = new Vector();
    Vector times = new Vector();
    Vector buttons = new Vector();
}
```

Times 可変長配列の確保

このように `AbstractDisplay` 基本クラスをドメインフレームワークの部品として登録し、モデルから生成されるコードに組み込まれるようにジェネレータを設定すれば、アプリケーションごとに個々に記述されるような処理も共通部品化することができます。結果としてコードジェネレータの定義は容易になり、また共通部品として再利用も可能となります (更なるソフトウェアプロダクトライン開発)

ご参考として最後に、この例に於けるディスプレイへの表示方法を説明します。確保された表示領域をディスプレイに表示する方法は、ターゲットプラットフォームによって様々でしょう。この例では表示領域が確保されると、その領域が一定時間ごとに JavaVM のプラットフォームによって更新されます。表示領域の更新が発生すると、フレームワーク内の以下のコードが呼ばれ、DisplayX234 オブジェクトの times を参照しながら、JavaVM 上の表示を更新します。times の各 Zone の内容を参照し、その値を JavaVM 特定の領域に描画しています。

